2015 IEEE 56th Annual Symposium on Foundations of Computer Science

Planar Reachability in Linear Space and Constant Time

Jacob Holm and Eva Rotenberg and Mikkel Thorup
Department of Computer Science, University of Copenhagen, Denmark,
jaho@di.ku.dk, roden@di.ku.dk, mthorup@di.ku.dk

Abstract

We show how to represent a planar digraph in linear space so that reachability queries can be answered in constant time. The data structure can be constructed in linear time. This representation of reachability is thus optimal in both time and space, and has optimal construction time. The previous best solution used $O(n \log n)$ space for constant query time [Thorup FOCS'01].

Keywords

Planar graphs, directed graphs, data structures, reachability.

I. Introduction

Representing reachability of a directed graph is a fundamental challenge. We want to represent a digraph $G = (V, E)$, $n = |V|$, $m = |E|$, so that we for any vertices u and v can tell if u reaches v, that is, if there is a dipath from u to v. There are two extreme solutions: one is to just store the graph, as is, using $O(m)$ words of space and answering reachability queries from scratch, e.g., using breadth-first-search, in $O(m)$ time. The other is to store a reachability matrix using n^2 bits and then answer reachability queries in constant time. Thorup and Zwick [21] proved that there are graphs classes such that any representation of reachability needs $\Omega(m)$ bits. Also, Pătraşcu [16] has proved that there are directed graphs with $O(n)$ edges where constant time reachability queries require $n^{1+\Omega(1)}$ space. Thus, for constant time reachability queries to a general digraph, all we know is that the worst-case space is somewhere between $\Omega(m + n^{1+\Omega(1)})$ and n^2 bits.

The situation is in stark contrast to the situation for undirected/symmetric graphs where we can trivially represent reachability queries on $O(n)$ space and constant time, simply by enumerating the connected components, and storing with each vertex the number of the component it belongs to. Then u reaches v if and only if the have the same component number.

In this paper we focus on the planar case, which feels particularly relevant when you live on a sphere. For planar digraphs it is already known that we can do much better than for general digraphs. Back in 2001, Thorup [20] presented a reachability oracle for planar digraphs using $O(n \lg n)$ space for constant query time, or linear space for $O(\log n)$ query time. In this paper, we present the first improvement; namely an $O(n)$ space reachability oracle that can answer reachability queries in constant time. Note that this bound is asymptotically optimal; even to distinguish between the subclass of directed paths of length n, we need $\Omega(n \log n)$ bits. Our oracle is constructed in linear time.

Computational model: The computational model for all upper bounds is the word RAM, modelling what we can program in a standard programming language such as C [12]. A word is a unit of space big enough to fit any vertex identifier, so a word has $w \geq \lg n$ bits, and word operations take constant time. Here $\lg = \log_2$. In our upper bounds, we limit ourselves to the *practical RAM* model [14], which is a restriction of the word RAM to the standard operations on words available in C that are AC^0. This includes indexing arrays as needed just to store a reachability matrix with constant time access, but excludes e.g. multiplication and division. Thus, unless otherwise specified, we measure *space* as the number of words used and *time* as the number of word operations performed.

The $\Omega(m + n^{1+\Omega(1)})$ space lower bound from [16] for general graphs is in the cell-probe model subsuming the word RAM with an arbitrary instruction set.

0272-5428/15 $31.00 © 2015 IEEE
DOI 10.1109/FOCS.2015.30

Other related work: Before [20], the best reachability oracles for general planar digraphs were distance oracles, telling not just if u reaches w, but if so, also the length of the shortest dipath from u to w [3]–[5]. For such planar distance oracles, the best current time-space trade-off is $\tilde{O}(n/\sqrt{s})$ time for any $s \in [n, n^2]$ [15].

The construction of [20] also yields approximate distance oracles for planar digraphs. With edge weights from $[N]$, $N \leq 2^w$, distance queries where answered within a factor $(1+\epsilon)$ in $O(\log\log(Nn) + 1/\varepsilon)$ time using $O(n(\log n)(\log(Nn))/\varepsilon)$ space. These bounds have not been improved.

For the simpler case of undirected graphs, where reachability is trivial, [13], [20] provides a more efficient $(1+\varepsilon)$-approximate distance queries for planar graphs in $O(1/\varepsilon)$ time and $O(n(\log n)/\varepsilon)$ space. In [10] it was shown that the space can be improved to linear if the query time is increased to $O((\log n)^2/\varepsilon^2)$. In [11] it was shown how to represent planar graphs with bounded weights using $O(n\log^2((\log n)/\varepsilon)\log^*(n)\log\log(1/\varepsilon))$ space and answering $(1+\varepsilon)$ approximate distance queries in $O((1/\varepsilon)\log(1/\varepsilon)\log\log(1/\varepsilon)\log^*(n)+\log\log\log n))$ time. Using $\bar{O}$ to suppress factors of $O(\log\log n)$ and $O(\log(1/\varepsilon))$, these bounds reduce to $\bar{O}(n)$ space and $\bar{O}(1/\varepsilon)$ time. This improvement is similar in spirit to our improvement for reachability in planar digraphs. However, the techniques are entirely different.

There has also been work on special classes of planar digraphs. In particular, for a planar s-t-graph, where all vertices are on dipaths between s and t, Tamassia and Tollis [18] have shown that we can represent reachability in linear space, answering reachability queries in constant time. Also, [4], [6], [7] present improved bounds for planar exact distance oracles when all the vertices are on the boundary of a small set of faces.

Techniques: We will develop our linear space constant query time reachability oracles by considering more and more complex classes of planar digraphs. We make reductions from $i+1$ to i in the following:

1) Acyclic planar s-t-graph; $\exists(s,t)$, such that all vertices are reachable from s and can reach t. [18]
2) Acyclic planar single-source graph; $\exists s$, such that all vertices are reachable from s. See Section III.
3) Acyclic planar In-Out graph; $\exists s$ such that all vertices with out-degree 0 are reachable from s. See Section IV
4) Any acyclic planar graph. The reduction to acyclic planar In-Out graphs from general acyclic planar graphs is known. [20]
5) Any planar graph. The reduction to acyclic planar graphs is well-known. Using the depth first search algorithm by Tarjan [19], we can contract each strongly connected component to get an acyclic planar graph. Vertices in the same strongly connected component can always reach each other, and vertices in distinct strongly connected components can reach each other if the corresponding vertices in the contracted graph can.

The most technically involved step is the reduction from single-source graph to s-t-graph. As in [20], we use separators to form a tree over a partitioning of the vertices of the graph. However, in [20], the *alternation number*; the number of directed segments in the frame that separates a child from its parent (see Section II), needs only be a constant number. In contrast, it is a crucial part of our construction that the alternation number, which must be even, is at most 4. Also, in our data structure, paths cannot go upward in the rooted tree, whereas there is no such restriction in [20]. These two features let us use a level ancestor -like algorithm to quickly calculate the best ≤ 4 vertices in a given tree-node that can reach a given vertex v. Each component is an s-t-graph, and v can be reached by some u in the ancestral component if and only if u can reach at least one of these best ≤ 4 vertices.

II. Preliminaries

For a vertex v at depth d in a rooted forest T and an integer $0 \leq i \leq d$, the i'th *level ancestor* of v in T is the ancestor to v in T at depth i. For two nodes x, y in a rooted tree, let $x \preceq y$ denote that x is an ancestor to y, and $x \prec y$ that x is a proper ancestor to y.

We say a graph is *plane*, if it is embedded in the plane, and denote by π_v the permutation of edges around v. Given a plane graph, (G, π), we may introduce *corners* to describe the incidence of a vertex to a face. A vertex of degree n has n corners, where if $\pi_v((v,u)) = (v,w)$, and the face f is incident to (v,u) and (v,w), then there is a corner of f incident to v between (v,u) and (v,w). We denote by $V[X]$ and $E[X]$ the vertices and edges, of some (not necessarily induced) subgraph X. Given a subgraph H of a planar embedded graph G, the faces of H define *superfaces* of those of G, and the faces of G are *subfaces* of those of H. Similarly for corners. Note that the faces of H correspond to the connected components of $G^* \setminus H$, where G^* is the dual graph of G. The super-corners incident to v correspond to a set of consecutive corners in the ordering around v.

In a directed graph, we may consider the boundary of a face in some subgraph, H. A corner of a face f of H is a *target* for f if it lies between ingoing edges (u,v) and (w,v), and *source* if it lies between outgoing edges (v,u) and (v,w). We say the face boundary has *alternation number* $2a$ if it has a source and a target corners. When a face boundary has alternation number $2a$, we say it consists of $2a$ *disegments* (*di*rected segments), associated with the directed paths from source to target. We associate with each disegment S the total ordering stemming from reachability of vertices on the path via the path, and by convention we set $\mathrm{succ}(t, S) = \perp$ for a target vertex t on the disegment. Given a set of edges $S \subset E$, we denote by $\mathrm{tail}(S)$ the set of inital vertices, $\mathrm{tail}(S) = \{u | (u,v) \in S\}$. Given a connected planar graph with a spanning tree T, the edges $T^* := E \setminus T$ form a spanning tree for the dual graph. We call the pair (T, T^*) a tree-cotree decomposition of the graph, referring to T and T^* as tree and cotree.

When u can reach v we write $u \rightsquigarrow v$. An s-t-graph is a graph with special vertices s, t such that $s \rightsquigarrow v$ and $v \rightsquigarrow t$ for all vertices v. We say a graph is a *truncated s-t-graph* if it is possible to add vertices s, t to obtain an s-t-graph, without violating the embedding. In an acyclic planar s-t-graph, all faces has alternation number 2 (see [17, Lemma 1]).

III. Acyclic planar single-source digraph

Given a global source vertex s for the planar digraph, we wish to make a data structure for reachability queries. We do this by reduction to the s-t-case. A rooted tree with truncated s-t-graphs as nodes is obtained by recursively choosing a face f wisely, letting vertices that can reach vertices on f belong to this node, and partitioning all other vertices among the descendants of this node. As we shall see in Section III-A, this can be done in such a way that we obtain logarithmic height and such that the border between a node and its ancestors is a cycle of alternation number at most 4. We call this the *frame* of the node.

We always choose the truncated s-t-graph maximally, such that once a path crosses a frame, it does not exit the frame again. Thus, for u to reach v, u has to lie in a component which is ancestral to that of v, and since the alternation number of any frame between those two component is at most 4, the path could always be chosen to use one of the at most 4 different "best" vertices for reaching v on that frame. Thus, the idea is to do something inspired by level ancestry to find those "best" vertices in u's component. We handle the case of frames with alternation number 2 in Section III-C. Frames with alternation number 4 are similar but more involved, and the details are found in Section III-D.

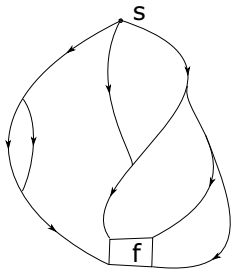

Figure 1. In a single source graph, the backwards closure of the face f is the union of all paths from the source, s, to $V[F]$.

Definition III.1. Given a graph $G = (V, E)$, a subgraph $G' = (V', E')$ is *backward closed* if $\forall (u,v) \in E : v \in V' \implies (u,v) \in E'$.

Definition III.2. The *backward closure* of a face f, denoted $\mathrm{bc}(f)$ is the unique smallest backward closed graph that contains all the vertices incident to f. (See Figure 1.)

Definition III.3. Let $G = (V, E)$ be an acyclic single-source plane digraph, and let $G^* = (V^*, E^*)$ be its dual. An *s-t-decomposition* of G is a rooted tree where each node x is associated with a face $f_x \in V^*$ and subgraphs $G^*_x \subseteq G^*$ and $C_x \subseteq S_x \subseteq G$ such that:

- f_x is unique ($f_x \neq f_y$ for $x \neq y$).
- S_x is $\mathrm{bc}(f_x)$ if x is the root, and $\mathrm{bc}(f_x) \cup S_y$ if x is a child of y.
- C_x is $\mathrm{bc}(f_x)$ if x is the root, and $\mathrm{bc}(f_x) \setminus S_y$ if x is a child of y.
- G^*_x is the subgraph of G^* induced by $\{f_z |\ z \text{ is a descendant of } x\}$. Furthermore, if x is a child of y we require that G^*_x is the connected component of $G^* \setminus E^*[S_y]$ containing f_x.

If x is a child of y, x has a *parent frame* $F_x \subseteq S_y$ and a set of *down-edges* $E_x \subseteq E$ such that:

- F_x is the face cycle in S_y that corresponds to G^*_x.
- E_x is the set of edges (w, w') such that $w \in V[F_x]$ and $w' \in V[C_z]$ for some descendant z of x.

An s-t-decomposition is *good* if the tree has height $\mathcal{O}(\log n)$ and each frame has alternation number 2 or 4.

That is, G^*_x is the set of faces contained in the parent frame F_x. The name s-t-decomposition is chosen based on the following

Lemma III.4. Each vertex of G is in exactly one C_x, and each C_x is a truncated s-t-graph.

Proof: If x is the root, $C_x = \mathrm{bc}(f_x)$ and this is clearly a truncted s-t-graph. Otherwise let y be the parent of x. Then $S_x = \mathrm{bc}(f_x) \cup S_y$, is backward-closed and therefore contains s. Contracting S_y in that graph to a single vertex s' gives a single-source graph S_x/S_y with s' as the source. Adding a target t' in f_x and edges (v, t') for $v \in V[f_x]$ results in an s-t-graph $(S_x/S_y) \cup \{t'\}$. Thus, S_x/S_y is a truncated s-t-graph, and since $C_x = \mathrm{bc}(f_x) \setminus S_y = S_x \setminus S_y = (S_x/S_y) \setminus \{s'\}$ so is C_x.

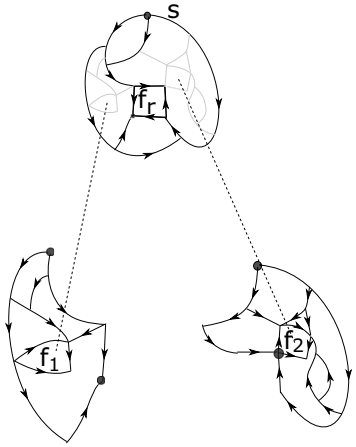

Figure 2. A tree of truncated s-t-graphs, each child contained in a face-cycle of its parent.

Let v be a vertex, let I be the set of all nodes in the s-t-decomposition whose associated faces $\{f_x\}_{x \in I}$ are reachable from v, and let $N = \mathrm{lca}(I)$. We now show that v lies in C_N and only in C_N. To see that $v \in C_N$, note that $v \in S_x$ for all $x \in I$, but then $v \in \bigcap_{x \in I} S_x = S_N$. But $v \notin S_a$ for any ancestor a of N by definition of lca, and thus, $v \notin S_y$ for the parent y of N, entailing $v \in S_N \setminus S_y = C_N$. We have now seen that $v \in C_N$ and that $v \notin C_x$ when $x \prec N$ or $N \prec x$. To see that $v \notin C_x$ for any unrelated $x \neq N$, note the following: if x has no descendants in I, then $v \notin V[S_x]$ since all vertices reachable from v lie on some face. Thus, $v \notin C_x \subseteq S_x$.

■

Theorem III.5. *Any acyclic single-source plane digraph has a good s-t-decomposition.*

We defer the proof to section III-A. The reason for studying s-t-decompositions in the context of reachability is the following

Lemma III.6. If $u \rightsquigarrow v$ where $u \in C_x$ and $v \in C_y$ then either $x = y$ or x has a child z that is ancestor of y such that any $u \rightsquigarrow v$ path contains a vertex in F_z.

Proof: Note that in general, whenever $w \rightsquigarrow w'$ with $w' \in C_a$, w must belong to an ancestor of a, since $w \in bc(f_a)$. Thus, in our case, x is an ancestor of y, which means that either $x = y$ or x has a child z that is an ancestor of y. But then either w lies on F_x, or F_x is a cycle separating w from w'. In either case, a path from w to w' must contain a vertex on F_x. ■

Since (by Theorem III.5) we can assume the alternation number is at most 4, this reduces the reachability question to the problem of finding the at most 4 "last" vertices on $F_z \cap C_x$ that can reach v and then checking in C_x if u can reach either of them. In section III-C we will show how to do this efficiently when F_z is *a* 2*-frame*, that is, has alternation number 2, and in section III-D we will extend this to the case when F_z is *a* 4*-frame*, that is, has alternation number 4.

Theorem III.7. *There exists a practical RAM data structure that for any planar digraph with n vertices uses $O(n)$ words of $O(\log n)$ bits and can answer reachability queries in constant time. The data structure can be built in linear time.*

Proof: First, build a good s-t-decomposition of G. Such a decomposition exists (Lemma III.5) and can be built in linear time (Lemma III.15). Adding DFS pre- and postorder numbers to each node in the tree lets us discover the ancestry relationship between any two vertices in the same node, in constant time.

To answer $\mathrm{reachable}(u, v)$, there are the following cases. Let $u \in C_x$ and $v \in C_y$ be the st-nodes of the st-decomposition where u and v belong, and let $\preceq$ denote ancestry of st-nodes in the st-decomposition.

1) If $x \not\preceq y$, then u cannot reach v.
2) If $x = y$, then the answer is given by the s-t-graph labelling of C_x from [18].
3) If $x \prec y$ and there are no 2-frames separating u and v, but, since $x \prec y$, there are 4-frames. Then, by Theorem III.46 we can in constant time compute the at most 4 best vertices in C_x that can reach v. If u can reach any of them, them u can reach v, otherwise no.
4) Otherwise, $x \prec y$ and there is a 2-frame F_z separating u and v such that there are no 2-frames between x and z. Then, by Theorem III.28 we can in constant time compute the at most 2 best vertices in C_z that can reach v. If u can reach any of them, then u can reach v, otherwise no.

Note that the recursive calls in step 3 only leads to questions of type 2, and similarly, the recursive calls in step 4 only leads to questions of type 3 or 2. Thus, we use only constant time per query. ■

A consequence of our construction which might be of independent interest is the following:

Theorem III.8. *If a planar digraph G admits an s-t-decompostion of height h where all frames have alternation number 2 and 4, there exists an $O(h \log n)$ bit labelling scheme for reachability with evaluation time $O(h)$.*

Especially, if a class of planar digraphs have such an s-t-decompositions of constant height, they have an $O(\log n)$ bit labelling scheme for reachability.

A. Constructing an s-t-decomposition

The s-t-decomposition recursively chooses a face f and consequently a subgraph $H = bc(f)$ of the graph G induced by all vertices that can reach a vertex on f. Since G was embedded in the plane, the subgraph H is embedded in the plane, and each vertex of $G \setminus H$ lies in a unique face of H. We may choose a tree-cotree decomposition wisely, such that for each face of H, the restriction of T^* to the subfaces of that face is again a dual spanning tree (Lemma III.10).

We also have to choose H carefully to ensure logarithmic height, and a limited alternation number on the frames. To ensure at most logarithmic height, we show two cases: 2-frame-nodes have only small children, while for 4-frame-nodes, we only need to ensure that their 4-frame children themselves are small.

Lemma III.9. Let $G = (V, E)$ be a plane graph, let $G^* = (V^*, E^*)$ be its dual, let (T, T^*) be a tree/cotree decomposition of G, and let S be a subgraph of G such that $S \cap T$ is connected. Then the faces of S correspond to connected components of $T^* \setminus E^*[S]$.

Proof: Let S^* be the dual of S, then $S^* = G^*/(G^* \setminus E^*[S])$ and the claim is equivalent to saying that the components of $G^* \setminus E^*[S]$ correspond to the components of $T^* \setminus E^*[S]$. Consider a pair of faces $f_1, f_2 \in V^*$. Clearly, if they are in separate components of $G^* \setminus E^*[S]$, they are also in separate components in $T^* \setminus E^*[S]$. On the other hand, suppose f_1 and f_2 are in different components in $T^* \setminus E^*[S]$. Then there exists an edge $e^* \in E^*[S] \cap T^*$ separating them. The corresponding edge $e \in E[S]$ induces a cycle in T, which is also part of S since $S \cap T$ is connected. The dual to that cycle is an edge cut in G^* that separates f_1 from f_2. ■

Lemma III.10. Let T be a spanning tree where all edges point away from the source s of G, then for any node x in an st-decomposition of G, the subgraph T_x^* of T^* induced by $V^*[G_x^*]$ is a connected subtree of T^*.

Proof: If x is the root, this trivially holds. If x has a parent y, G_x^* corresponds to a face in S_y. Now $S_y \cap T$ is connected since S_y is the union of backward-closed graphs, and the result follows from Lemma III.9. ■

Lemma III.11. Let x be a node in an st-decomposition whose parent frame F_x has alternation number 2, and let A^* be the set of faces in T_x^* incident to the target corner of F_x. Then for any child y of x:

$$A^* \subseteq V^*[T_y^*] \implies F_y \text{ has alternation number } 4.$$
$$A^* \not\subseteq V^*[T_y^*] \implies F_y \text{ has alternation number } 2.$$

Proof: Let t_x be the target corner of F_x and let A^* be the set of faces in ${T_x}^*$ incident to t_x. For any child y of x, F_y consists of a (possibly empty) segment of F_x and two directed paths that meet at a new target corner t_y. Each target corner of F_y must therefore be at either t_x or t_y. Now if $A^* \subseteq V^*[T_y^*]$, then both t_x and t_y are target corners of F_y, otherwise only t_y is. Either way the result follows. ■

The following lemma will help us ensure that we can choose all frames to be 2- or 4-frames.

Lemma III.12. Let x be a node in an st-decomposition whose parent frame F_x has alternation number 4, and let ${A^0}^*$ and ${A^1}^*$ be the sets of faces in T_x^* incident to the target corners of F_x. Then for any child y of x:

$${A^0}^* \not\subseteq V^*[T_y^*] \wedge {A^1}^* \not\subseteq V^*[T_y^*] \implies F_y \text{ has alternation number } 2.$$
$${A^0}^* \not\subseteq V^*[T_y^*] \vee {A^1}^* \not\subseteq V^*[T_y^*] \implies F_y \text{ has alternation number at most } 4.$$

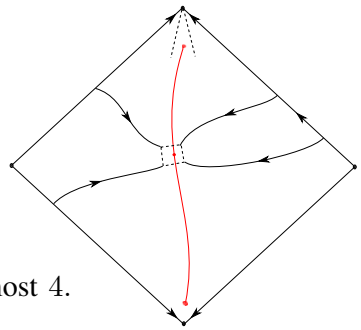

Figure 3. Choosing a splitting-face on the co-path between faces near either target of a 4-frames yields only 4-frames and 2-frames on the next level.

Proof: Let t_x^0 and t_x^1 be the two target corners of F_x and for $i \in \{0, 1\}$ let ${A^i}^*$ be the set of faces in ${T_x}^*$ incident to t_x^i. For any child y of x, F_y consists of a (possibly empty) segment of F_x and two directed paths that meet at a new target corner t_y. Each target corner of F_y must therefore be at either t_y, t_x^0, or t_x^1. Now if ${A^i}^* \not\subseteq V^*[T_y^*]$ for some $i \in \{0, 1\}$, then t_x^i is not a target corner of F_y. So the number of target corners in F_y is at least 1, and at most 3 minus the number of such i, and the result follows. ■

We can now prove that a good s-t-decomposition exists.

proof of theorem III.5: Let s be the source of G and let (T, T^*) be a tree/cotree decomposition of G such that all edges in T point away from s. The st-decomposition can be constructed recursively as follows. Start with the root. In each step we have a node x and by Lemma III.10 the subgraph T_x^* induced in T^* by $V^*[G_x^*]$ is a tree. The goal is to select a face f_x such that for each child y:

- The alternation number of F_y is at most 4, and
- For each child z of y (and thus grandchild of x), $|T_z^*| \leq \frac{1}{2}|T_x^*|$.

If we can do this for all x, we are done. There are 3 cases:

x is the root: Let f_x be the median of $T_x^* = T^*$. Then for each child y, $|T_y^*| \leq \frac{1}{2}|T_x^*|$, and, since $S_x = \mathrm{bc}(f_x)$ is a truncated s-t-graph with a single source, F_y has alternation number 2.

F_x has alternation number 2: Let f_x be the median of T_x^*. Then for each child y, $|T_y^*| \leq \frac{1}{2}|T_x^*|$, and, by Lemma III.11, F_y has alternation number at most 4.

F_x has alternation number 4: Let t_0 and t_1 be the local targets of F_x and let $f_0, f_1 \in V^*[T_x^*]$ be (not necessarily distinct) faces incident to t_0 and t_1 respectively. Now choose f_x as the projection of the median m of T_x^* on the path $f_0, \ldots, f_1$ in T_x^* (see Figure 3). By Lemma III.12 this means that for any child y of x, the alternation number of the parent frame F_y is at most 4.
- If $f_x = m$ then $|T_y^*| \leq \frac{1}{2}|T_x^*|$.
- If $f_x \neq m$ and T_y^* is not the component of m in $T_x^* \setminus E^*[\mathrm{bc}(f_x)]$, then $|T_y^*| \leq \frac{1}{2}|T_x^*|$.
- If $f_x \neq m$, and T_y^* is the component of m in $T_x^* \setminus E^*[\mathrm{bc}(f_x)]$, then T_y^* contains neither f_0 nor f_1, so by Lemma III.12 the parent frame F_y has alternation number at most 2 and we have just shown this means any child z of y has $|T_z^*| \leq \frac{1}{2}|T_y^*| \leq \frac{1}{2}|T_x^*|$. ■

B. Constructing a good s-t-decomposition in linear time

In the construction of an s-t-decomposition, a face is chosen, some edges are deleted, and new connected components of the dual graph arise. We then recurse on the new connected components of the dual graph. By Lemma III.10 we can choose a tree/cotree-decomposition such that each component that arises is spanned by a subtree of the cotree.

To obtain linear construction time, we use a variation of the decremental tree connectivity algorithm from [2] to keep track of the subtrees of the cotree, and associate some information with each subtree. In particular, when T_x^* is a component at some point, we can in constant time find the node x.

For each node x we keep the set of target vertices on F_x (or $\emptyset$ if x is the root), and a face in T_x^* incident to each target in the set.

Build a top tree (see [1]) of height $O(\log n)$ over T^*, and let v_{n-i}^* be the i'th face that stops being boundary during the construction. Using this enumeration, the boundary faces of a cluster will be visited before boundary faces of their descendants. We use this ordering to find the splitting faces of the s-t-decomposition.

For each v_i^*, we can use the connectivity structure to find the relevant node x to split. We then need to choose the target face f_x defining the split. If x is the root or F_x is a 2-frame, we just set $f_x = v_i^*$. If F_x is a 4-frame, the information in x contains a pair of faces f_1, f_2 and we use a static nearest common ancestor data structure from Harel and Tarjan [8] to find the projection $f_x = \pi(v_i^*)$ of v_i^* on $f_1, \cdots, f_2$. Note that the projection of v_i^* is always contained in the same connected component as f_1, f_2, and thus, the data structure for the whole tree suffices to answer this query for the particular subtree.

Once f_x has been selected, we traverse the graph backwards from the vertices of f_x until we have found all the edges with destination in C_x. This search takes $|C_x|$ time. We delete these edges from the forest as we go along. Once we are done, we take all targets in C_x and select an incident face for each component it is incident to. This again takes $|C_x|$ time. If $f_x \neq v_i^*$ we try with v_i^* again, otherwise we move on to v_{i+1}^*.

Lemma III.13. The s-t-decomposition constructed via the approach sketched above has no frame of alternation number > 4.

Proof: Components with 2-frames always have children with 2- and 4-frames. For components with 4-frames, this follows directly from Lemma III.11, since we chose a splitting face on the cotree path between faces near the two targets. ■

Lemma III.14. The s-t-decomposition constructed via the approach sketched above has height $O(\log n)$.

Proof: Since the top-tree has height $O(\log n)$, choosing the boundary face v_i^* as a splitting face every time would result in a tree of the same height; $O(\log n)$. However, for each 4-frame, we might

choose a face $f_x \neq v_i^*$ which is the projection of v_i^* on $f_1 \dots f_2$. As noted in Lemma III.11, when this happens, v_i^* will lie in a child which has a 2-frame. But then, v_i^* will be the splitting face for that child. We thus increase the height by no more than a factor 2, and the s-t-decomposition has height $2O(\log n) = O(\log n)$. ■

Lemma III.15. Let $G = (V, E)$ be a plane single-source graph with source s, then we can construct a good s-t-decomposition of G in linear time.

Proof: Since the top-tree can be constructed in linear time, and since the decremental connectivity for trees takes linear time, and since the static nearest common ancestor data stucture is constructed in linear time and answers queries in constant time, the construction takes linear time. By Lemma III.13 and III.14, the resulting s-t-decomposition is good. ■

C. 2-frames

Given a graph G with vertices V and edges E, we have shown how to construct an st-decomposition $\mathcal{T}$ in linear time. An st-decomposition is a tree with st-nodes and st-edges. Each non-root st-node, x, has a frame F_x, which will either be a 2-frame or a 4-frame. We say that the st-node has a 2- or a 4-frame.

Since we want to reason about the relationship between 2-frames, disregarding 4-frames, we construct a 2-*frame-decomposition* from the st-decomposition. The 2-frame-decomposition is a tree whose nodes are contracted subtrees of the st-decomposition. Specifically, each st-node with a 4-frame is contracted with its nearest ancestor with a 2-frame.

Definition III.16. Let $\mathcal{T}$ be an st-decomposition of $G = (V, E)$. Then we can define a 2-*frame-decomposition* $\mathcal{T}_2$ by contracting each st-edge (z, y) in $\mathcal{T}$ where the child z of y has a 4-frame. For each node x in $\mathcal{T}_2$ that is contracted from a set of nodes $Y \subseteq \mathcal{T}$, define $C_x := \bigcup_{y \in Y} C_y$, and if x is not the root, define $F_x := F_{\text{lca}(Y)}$ and $E_x := E_{\text{lca}(Y)}$.

Recall, E_x are the edges with their tail on the frame F_x. If the frame is a 2-frame, it consists of a clockwise and a counterclockwise disegment. The embedding of G gives a natural cyclic order to E_x, and we can partition the edges of E_x into two contiguous subsets in that cyclic order, such that all tails in one subset are on the clockwise disegment of F_x and all tails in the other subset are on the counterclockwise disegment of F_x. We notice that there exists a partitioning of all edges whose tail is on a 2-frame, into sets $\mathcal{R}$ and $\mathcal{L}$ such that for *any* E_x, the sets $E_x \cap \mathcal{L}$ and $E_x \cap \mathcal{R}$ form such a partition.

Definition III.17. Let $(\mathcal{L}, \mathcal{R})$ be the partition of $\cup_{x \in \mathcal{T}_2} E_x$ defined as follows: For each $(u, v) \in \cup_{x \in \mathcal{T}_2} E_x$ let y be the node (if it exists) closest to the root of $\mathcal{T}_2$ such that $(u, v) \in E_y$ but u is not the target vertex of F_y. If y exists and (u, v) is incident to a corner on the clockwise disegment of F_y between s_y and t_y assign (u, v) to $\mathcal{R}$, otherwise assign (u, v) to $\mathcal{L}$.

Lemma III.18. Let $(\mathcal{L}, \mathcal{R})$ be the partition from Definition III.17. Then for any $x \in \mathcal{T}_2$, $(E_x \cap \mathcal{L}, E_x \cap \mathcal{R})$ is a partition of E_x, such that all tails in $E_x \cap \mathcal{L}$ are on the clockwise disegment of F_x, and all tails in $E_x \cap \mathcal{R}$ are on the counterclockwise disegment of F_x.

Proof: Let $x \in \mathcal{T}_2$, let $(u, v) \in E_x$. If u is the source or the target of F_x, it is on both the disegments, and we are done. Suppose therefore that u is neither, and thus, lies on exactly one disegment of F_x. Let $y \in \mathcal{T}_2$ be the st-node closest to the root such that $(u, v) \in E_y$, and such that u is not the target vertex of E_y, as in the definition. Since u is not the source vertex of F_x, u cannot be the source vertex of F_y. That means, u lies on exactly one disegment of F_y.

Assume for contradiction that u lies on the clockwise disegment of F_x but the counterclockwise disegment of F_y. Then, there must be some st-node x' on the path between y and x in $\mathcal{T}_2$ such that u lies on the clockwise disegment of x' and on the counterclockwise disegment of x''s parent, z.

First note that u cannot be the target vertex of F_z, since then $x = x' = y$. Also, u cannot be the source of neither F_z or $F_{x'}$, as then u would be the source of F_x. So u lies only on the counterclockwise disegment of F_z. Then, the source vertex $s_{x'}$ of $F_{x'}$ belongs to $bc(f_z)$, and any non-trivial path from $s_{x'}$ to u would belong to C_z, and thus to u's frame at that level, causing (u, v) to belong to the clockwise disegment. That means if u belongs to the counterclockwise disegment of F_z, we must have $s_{x'} = u$, a contradiction. ■

Each vertex belongs to some node of $\mathcal{T}_2$, and thus, we can define the depth of the vertex in the $\mathcal{T}_2$-tree:

Definition III.19. Let $\mathcal{T}_2$ be a 2-frame-decomposition of $G = (V, E)$. For any vertex $v \in V$ define:

$$c_2[v] := \text{The node } x \text{ in } \mathcal{T}_2 \text{ such that } v \in V[C_x]$$
$$d_2[v] := \text{The depth of } c_2[v] \text{ in } \mathcal{T}_2$$

Given a vertex, v, and given a frame F that is ancestral to v in the st-decomposition, each disegment on F contains a last vertex that can reach v via an ingoing edge. Thus, all other vertices on the frame can reach v via an ingoing edge if and only if they can reach v via one of those last vertices (see Figure 4). For 2-frames, such vertices will be called $l_i(v)$ and $r_i(v)$ and are defined as follows:

Definition III.20. For any $0 \leq i < d_2[v]$, let x be the ancestor of $c_2[v]$ at depth $i + 1$ and define:

$$E_i(v) := E_x$$
$$L_i(v) := E_x \cap \mathcal{L}$$
$$R_i(v) := E_x \cap \mathcal{R}$$
$$\widehat{L}_i(v) := \left\{(w, w') \in L_i(v) \middle| \; w' \rightsquigarrow v\right\}$$
$$\widehat{R}_i(v) := \left\{(w, w') \in R_i(v) \middle| \; w' \rightsquigarrow v\right\}$$
$$\widehat{F}_i(v) := \widehat{L}_i(v) \cup \widehat{R}_i(v)$$
$$l_i(v) := \begin{cases} \perp & \text{if } \widehat{L}_i(v) = \emptyset \\ \text{the last vertex in } \mathrm{tail}(\widehat{L}_i(v)) \text{ on the counterclockwise dipath of } F_x & \text{otherwise} \end{cases}$$
$$r_i(v) := \begin{cases} \perp & \text{if } \widehat{R}_i(v) = \emptyset \\ \text{the last vertex in } \mathrm{tail}(\widehat{R}_i(v)) \text{ on the clockwise dipath of } F_x & \text{otherwise} \end{cases}$$

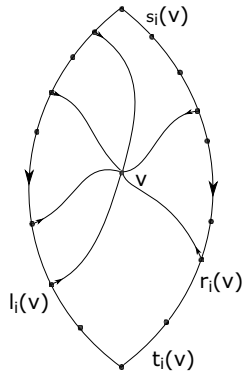

Figure 4. The best two vertices that can reach v on level i.

Additionally, let $L_i(v)$ and $\widehat{L}_i(v)$ be totally ordered by the position of the starting vertices on the counterclockwise disegment of F_x and the clockwise order around each starting vertex. Similarly let $R_i(v)$ and $\widehat{R}_i(v)$ be totally ordered by the position of the starting vertices on the clockwise disegment of F_x and the counterclockwise order around each starting vertex.

The goal in this section is a data structure for efficiently computing $l_i(v)$ and $r_i(v)$ for $0 \leq i < d_2[v]$. The main idea that *almost* works is to represent each function with a suitable rooted forest and use a level ancestor structure on that forest to answer queries.

Definition III.21. For any vertex $v \in V$, define the left parent $p_l[v]$ and right parent $p_r[v]$ as

$$p_l[v] := \begin{cases} \perp & \text{if } d_2[v] = 0 \\ l_{d_2[v]-1}(v) & \text{otherwise} \end{cases} \qquad p_r[v] := \begin{cases} \perp & \text{if } d_2[v] = 0 \\ r_{d_2[v]-1}(v) & \text{otherwise} \end{cases}$$

and let T_l and T_r denote the rooted forests over V whose parent pointers are p_l and p_r respectively.

Using these trees we can define functions $l'_i(v)$ and $r'_i(v)$ (related but not always equal to $l_i(v)$ and $r_i(v)$), as the nearest ancestor to v in T_l or T_r with depth $\leq i$. We will later show how to use a level ancestor structure to compute these efficiently, but for our proofs it is more convenient to define them recursively as follows.

Definition III.22. For any $v \in V \cup \{\perp\}$, and $i \geq 0$ let

$$l'_i(v) := \begin{cases} v & \text{if } v = \perp \vee d_2[v] \leq i \\ l'_i(p_l[v]) & \text{otherwise} \end{cases} \qquad r'_i(v) := \begin{cases} v & \text{if } v = \perp \vee d_2[v] \leq i \\ r'_i(p_r[v]) & \text{otherwise} \end{cases}$$

As mentioned, we do not always have $l_i(v) = l'_i(v)$ and $r_i(v) = r'_i(v)$. When F_y is the parent frame of F_x in $\mathcal{T}_2$, there are two cases for the best vertices on F_y in relation to the best vertices of F_x, call them l, r. Either the left parent of l and the right parent of r are the best vertices on F_y, or there is a *crossing*, to one or the other side, e.g. when the left parent of r is later than the left parent of l on their disegment of F_y (see Figure 5).

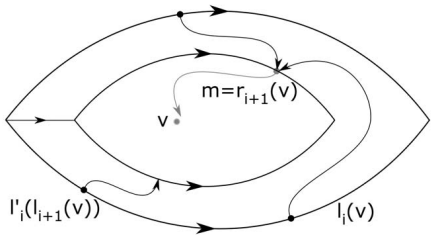

Figure 5. The best path from $L_i(v)$ to v goes via $r_{i+1}(v)$.

Lemma III.23 (Crossing lemma)**.** Let $v \in V$, and $0 \leq i < d_2[v] - 1$.

$$\begin{aligned} l_i(v) \neq l'_i(l_{i+1}(v)) \implies & \quad l_i(v) = l'_i(m) \wedge r_i(v) = r'_i(m) \wedge d_2[m] = i+1 \\ & \text{where } m = r_{i+1}(v) \neq \perp \\ r_i(v) \neq r'_i(r_{i+1}(v)) \implies & \quad l_i(v) = l'_i(m) \wedge r_i(v) = r'_i(m) \wedge d_2[m] = i+1 \\ & \text{where } m = l_{i+1}(v) \neq \perp \end{aligned}$$

The proof needs som additional technical lemmas, and is deferred to the end of this section. What the lemma says is that when there is a crossing at level i, then there exists some *crossing vertex* $m \in \{r_{i+1}(v), l_{i+1}(v)\}$ such that the left- and right-parents of m are the best vertices that can reach v on level i. We define $m_i(v)$ as either the crossing vertex at level i, if there is a crossing, or the nearest crossing vertex on a descendent level, otherwise.

Definition III.24. Let $v \in V$ and $0 \leq i < d_2[v]$.

$$m_i(v) := \begin{cases} v & \text{if } i+1 = d_2[v] \\ l_{i+1}(v) & \text{if } i+1 < d_2[v] \wedge r_i(v) \neq r'_i(r_{i+1}(v)) \\ r_{i+1}(v) & \text{if } i+1 < d_2[v] \wedge l_i(v) \neq l'_i(l_{i+1}(v)) \\ m_{i+1}(v) & \text{otherwise} \end{cases}$$

Corollary III.25. Let $v \in V$ and $0 \leq i < d_2[v] - 1$. If $l_i(v) \neq l'_i(l_{i+1}(v))$ or $r_i(v) \neq r'_i(r_{i+1}(v))$ then

$$l_i(v) = l'_i(m_i(v)) \qquad \wedge \qquad r_i(v) = r'_i(m_i(v)) \qquad \wedge \qquad d_2[m_i(v)] = i+1$$

Proof: This is just a reformulation of Lemma III.23 in terms of $m_i(v)$. ∎

Given this definition of $m_i(v)$, one may always find the best vertices that can reach v, $l_i(v)$ and $r_i(v)$ as the left- and right-ancestors of $m_i(v)$.

Lemma III.26. For any vertex $v \in V$ and $0 \leq i < d_2[v]$

$$l_i(v) = l'_i(m_i(v)) \qquad \wedge \qquad r_i(v) = r'_i(m_i(v))$$

Proof at the end of this section.

To calculate $l_i(v)$ and $r_i(v)$ quickly for some given i, the idea is to store for each vertex a bit array of whether there is a crossing at a given level, then to calculate the crossing vertex for that level, and then find the left- and right-ancestors for that crossing vertex. To represent the function $m_i(v)$, we again use a suitable rooted forest, and use a level ancestor structure on that forest to answer queries.

Definition III.27. For any vertex $v \in V$, let

$$M[v] := \{i |\ 0 < i < d_2[v] \wedge m_{i-1}(v) \neq m_i(v)\}$$
$$p_m[v] := \begin{cases} \perp & \text{if } M[v] = \emptyset \\ m_{\max M[v]-1}(v) & \text{otherwise} \end{cases}$$

And define T_m as the rooted forest over V whose parent pointers are p_m.

Theorem III.28. *There exists a practical RAM data structure that for any good st-decomposition of a graph with n vertices uses $\mathcal{O}(n)$ words of $\mathcal{O}(\log n)$ bits and can answer $l_i(v)$ and $r_i(v)$ queries in constant time.*

Proof: For any vertex $v \in V$, let

$$D_l[v] := \{i |\ v \text{ has a proper ancestor } w \text{ in } T_l \text{ with } d_2[w] = i\}$$
$$D_r[v] := \{i |\ v \text{ has a proper ancestor } w \text{ in } T_r \text{ with } d_2[w] = i\}$$

Now, store level ancestor structures for each of T_l, T_r, and T_m, together with $d_2[v]$, $D_l[v]$, $D_r[v]$, and $M[v]$ for each vertex. Since the height of the st-decomposition is $\mathcal{O}(\log n)$ each of $D_l[v]$, $D_r[v]$, and $M[v]$ can be represented in a single $\mathcal{O}(\log n)$-bit word.

This representation allows us to find $d_2[m_i(v)] = \operatorname{succ}(M[v] \cup \{d_2[v]\}, i)$ in constant time, as well as computing the depth in T_m of $m_i(v)$. Then, using the level ancestor structure for T_m, we can compute $m_i(v)$ in constant time.

Similarly, this representation of the $D_l[v]$ set lets us compute the depth in T_l of $l'_i(v)$ in constant time, and with the level ancestor structure, this lets us compute $l'_i(v)$ in constant time. A symmetric argument shows that we can compute $r'_i(v)$ in constant time.

Finally, lemma III.26 says we can compute $l_i(v)$ and $r_i(v)$ in constant time given constant-time functions for l', r', and m. ∎

Technical lemmas and proofs: To prove Lemmas III.23 and III.26, we use some technical lemmas.

Lemma III.29. For any $u, v \in V$ and $0 \leq i < d_2[u]$: If $u \rightsquigarrow v$ then $\widehat{L}_i(u) \subseteq \widehat{L}_i(v)$ and $\widehat{R}_i(u) \subseteq \widehat{R}_i(v)$.

Proof: Since $u \rightsquigarrow v$, $c_2[u]$ is ancestor to $c_2[v]$ and so $L_i(u) = L_i(v)$ and hence $\widehat{L}_i(u) \subseteq \widehat{L}_i(v)$. Similarly, $R_i(u) = R_i(v)$ and $\widehat{R}_i(u) \subseteq \widehat{R}_i(v)$. ∎

When an edge (w, w') that lies on a path from s to v skips several levels, that is, w' lies on a level possibly much higher than that of w, then (w, w') belongs to either $\hat{L}$ or $\hat{R}$ of v for all those levels:

Lemma III.30. Given any vertex $v \in V$, $0 \leq i < d_2[v]$, and $(w, w') \in E_i(v)$. Then:

$$(w, w') \in \widehat{L}_i(v) \implies (w, w') \in \widehat{L}_{i'}(v) \text{ for all } i', d_2[w] \leq i' < \min\left\{d_2[w'], d_2[v]\right\}$$
$$(w, w') \in \widehat{R}_i(v) \implies (w, w') \in \widehat{R}_{i'}(v) \text{ for all } i', d_2[w] \leq i' < \min\left\{d_2[w'], d_2[v]\right\}$$

Proof: Let $j = d_2[w]$ and $k = \min\{d_2[w'], d_2[v]\}$. Clearly $(w, w') \in E_{i'}(v)$ for all $j \leq i' < k$. Suppose $(w, w') \in \widehat{L}_i(v) \subseteq L_i(v)$, then since $j \leq i < k$ the definition give us $(w, w') \in L_{i'}(v)$ for all $j \leq i' < k$. And since $w' \rightsquigarrow v$ this implies $(w, w') \in \widehat{L}_{i'}(v)$ for all $j \leq i' < k$ and the result follows. The case for R is symmetric. ∎

The vertices that are left and right level ancestors $l'_i(v)$ and $r'_i(v)$ from Definition III.22 are tails of edges of $\hat{L}_i(v)$ and $\hat{R}_i(v)$:

Lemma III.31. Let $v \in V$, and $i \geq 0$ be given, then

$$\begin{array}{lllll} i = d_2[v] - 1 & \implies & l'_i(v) = l_i(v) & \wedge & r'_i(v) = r_i(v) \\ i \leq d_2[v] - 1 & \implies & l'_i(v) \in \text{tail}(\widehat{L}_i(v)) \cup \{\bot\} & \wedge & r'_i(v) \in \text{tail}(\widehat{R}_i(v)) \cup \{\bot\} \\ i > d_2[v] - 1 & \implies & l'_i(v) = v & \wedge & r'_i(v) = v \end{array}$$

Proof: We will show this for l' only, as r' is completely symmetrical. If $i > d_2[v]-1$ then $d_2[v] \leq i$ and we get $l'_i(v) = v$ directly from the definition of l'. Similarly if $i = d_2[v]-1$ then $l'_i(v) = l'_i(p_l[v]) = l'_i(l_{d_2[v]-1}(v)) = l'_i(l_i(v)) = l_i(v) \in \text{tail}(\widehat{L}_i(v)) \cup \{\bot\}$. Finally suppose $i < d_2[v]-1$. If $l'_i(v) = \bot$ we are done, so suppose that is not the case. Let u be the child of $l'_i(v)$ in T_l that is ancestor to v. Then $l'_i(v) = l'_i(u) = p_l[u] = l_{d_2[u]-1}(u)$. By definition of $l_{d_2[u]-1}(u)$ there exists an edge $(w, w') \in \widehat{L}_{d_2[u]-1}$ where $w = l_{d_2[u]-1}(u)$ and $d_2[w] \leq i < d_2[w'] \leq d_2[u]$ and by setting $(v, i, (w, w')) = (u, d_2[u] - 1, (w, w'))$ in Lemma III.30 we get $(w, w') \in \widehat{L}_i(u)$, and therefore $l'_i(v) \in \text{tail}(\widehat{L}_i(u))$. But since $u \rightsquigarrow v$ we have $\widehat{L}_i(u) \subseteq \widehat{L}_i(v)$ by Lemma III.29 and we are done. ∎

We prove the following two intuitive lemmas: The level ancestor of a level ancestor of v is again the level ancestor of v, and if $l_i(v) = \bot$, then the i'th level ancestor of $l_{i+1}(v)$ is also $\bot$ (similar for $r_i(v)$).

Lemma III.32. Let $v \in V$ and $0 \leq i \leq j$ then

$$l'_i(l'_j(v)) = l'_i(v) \qquad \wedge \qquad r'_i(r'_j(v)) = r'_i(v)$$

Proof: $l'_j(v)$ is on the path from v to $l'_i(v)$ in T_l, so this follows trivially from recursion. The case for r' is symmetric. ∎

Lemma III.33. Let $v \in V$, and $0 \leq i < d_2[v] - 1$, then

$$l_i(v) = \bot \implies l'_i(l_{i+1}(v)) = \bot \quad \wedge \quad r_i(v) = \bot \implies r'_i(r_{i+1}(v)) = \bot$$

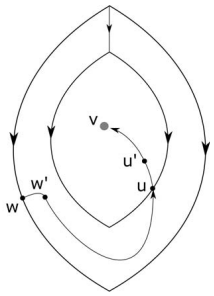

Figure 6. If $l'_i(l_{i+1}) \neq l_i$, then any path from l_i to v must go through $R_{i+1}(v)$.

Proof: If $l_i(v) = \bot$ then $\widehat{L}_i(v) = \emptyset$, so either $l_{i+1}(v) = \bot$ implying $l'_i(l_{i+1}(v)) = \bot$ by the definition of l', or $l_{i+1}(v) \notin \text{tail}(\widehat{L}_i(v))$ so $d_2[l_{i+1}(v)] = i + 1$ and by Lemma III.31 and Lemma III.29 $l'_i(l_{i+1}(v)) \in \text{tail}(\widehat{L}_i(l_{i+1}(v))) \cup \{\bot\} \subseteq \text{tail}(\widehat{L}_i(v)) \cup \{\bot\} = \{\bot\}$ so again $l'_i(l_{i+1}(v)) = \bot$. The case for r is symmetric. ∎

We may now prove Lemma III.23 (Crossing Lemma).

Proof of Lemma III.23 (Crossing Lemma): Suppose $l_i(v) \neq l'_i(l_{i+1}(v))$ (the case $r_i(v) \neq r'_i(r_{i+1}(v))$ is symmetrical). Let $m = r_{i+1}(v)$. We then aim to show, first, $d_2[m] = i+1$, and secondly, $l_i(v) = l'_i(m)$ and $r_i(v) = r'_i(m)$. If $l_i(v) \neq l'_i(l_{i+1}(v))$, then $l_i(v) \neq \bot$ by lemma III.33. Thus, there is a last edge $(w, w') \in \widehat{L}_i(v)$ with $w = l_i(v)$ and $d_2[w] \leq i < d_2[w']$ and a path $P = w' \rightsquigarrow v$.

Now, $(w, w') \notin E_{i+1}(v)$, since otherwise by Definition III.20 $(w, w') \in L_{i+1}(v)$ and since $w' \rightsquigarrow v$ even $(w, w') \in \widehat{L}_{i+1}(v)$ implying $l_i(v) = l_{i+1}(v)$ and thus $l_i(v) = l'_i(l_{i+1}(v))$ by Lemma III.31, contradicting our assumption.

Since $(w, w') \notin E_{i+1}(v)$, the path P must cross $\widehat{F}_{i+1}(v)$. (See Figure 6.) Let (u, u') be the unique edge in $P \cap \widehat{F}_{i+1}(v)$. Then, $w' \rightsquigarrow u$ so $d_2[u] \geq d_2[w'] \geq i + 1$ and $(u, u') \notin L_{i+1}(v)$ since otherwise $d_2[l_{i+1}(v)] = i + 1$ and hence by Lemma III.31 $l_i(v) = l'_i(l_{i+1}(v))$, again contradicting our assumption. Since $\widehat{F}_{i+1}(v) \neq \emptyset$, we therefore have $(u, u') \in \widehat{R}_{i+1}(v)$. But then we can choose P so it goes through (m, m') where $m = r_{i+1}(v) \neq \bot$. Now $i + 1 \leq d_2[w'] \leq d_2[r_{i+1}(v)] \leq i + 1$ so $d_2[m] = i + 1$.

Finally, let e be the last edge in $\widehat{R}_i(v)$. Then, any path $r_i(v) \rightsquigarrow v$ that starts with e crosses $P \cup \widehat{R}_{i+1}(v)$, implying that there exists such a path that contains (m, m') and thus $r_i(v) = r_i(m)$. Since $d_2[m] = i+1$, then $l_i(v) = l'_i(m)$ and $r_i(v) = r'_i(m)$ follows from Lemma III.31. ∎

We may now prove the essential Lemma stating that $l_i(v) = l'_i(m_i(v))$ (and similar for r_i):

Proof of Lemma III.26: The proof is by induction on j, the number of times the "otherwise" case is used before reaching one of the other cases when expanding the recursive definition of $m_i(v)$.

For $j = 0$, either $i+1 = d_2[v]$ and the result follows from Lemma III.31, or $i+1 < d_2[v]$ and $l_i(v) \neq l'_i(l_{i+1}(v))$ or $r_i(v) \neq r'_i(r_{i+1}(v))$. In either case we have by Corollary III.25 that $l_i(v) = l'_i(m_i(v))$ and $r_i(v) = r'_i(m_i(v))$.

For $j > 0$ we have $i+1 < d_2[v]$ and $l_i(v) = l'_i(l_{i+1}(v))$ and $r_i(v) = r'_i(r_{i+1}(v))$ and $m_i(v) = m_{i+1}(v)$. By induction we can assume that $l_{i+1}(v) = l'_{i+1}(m_{i+1}(v))$ and $r_{i+1}(v) = r'_{i+1}(m_{i+1}(v))$. Then by Lemma III.32, $l'_i(l_{i+1}(v)) = l'_i(l'_{i+1}(m_{i+1}(v))) = l'_i(m_{i+1}(v)) = l'_i(m_i(v))$, showing that $l_i(v) = l'_i(m_i(v))$ as desired. The case for r is symmetric. ∎

D. 4-frames

Given a vertex and an ancestral frame, F, we have now seen how to find the at most two best vertices on F that can reach v when F a 2-frame. A similar statement is true for 4-frames, only now we have four disegments, and thus, up to four best vertices. To find the best vertices, we may use Theorem III.28 as a subroutine, and thus we only need to get from the nearest descendent 2-frame, if it exists, to a given level. That is, we may disregard edges that cross a 2-frame.

In our construction, we exploit that whenever a 4-frame occurs, it shares at least one target with its parent frame. In particular, if its parent is again a 4-frame, they share a target vertex.

Definition III.34. Let x be a node in an s-t-decomposition such that F_x is a 4-frame, and let y be its parent. Let s_x^0 and s_x^1 be the source corners on F_x and let t_x^0 and t_x^1 be the target corners on F_x, numbered such that their clockwise cyclic order on F_x is $s_x^0, t_x^0, s_x^1, t_x^1$, and such that if F_y is a 4-frame there is an $\alpha \in \{0,1\}$ so $t_x^\alpha = t_y^\alpha$.

Recall from 2-frames that we found a global partition of all the down-edges whose tail is on a 2-frame into two sets $\mathcal{L}$ and $\mathcal{R}$. It turns out we can partition the remaining down-edges (those that are only in 4-frames) into four sets $\mathcal{L}^0$, $\mathcal{R}^0$, $\mathcal{L}^1$, and $\mathcal{R}^1$, such that for edges that do not cross a 2-frame, the partitioning corresponds to the four disegments.

Definition III.35. Let $\mathcal{E}_4$ be the set of down-edges (u, v) such that for all st-nodes x where $(u, v) \in E_x$, F_x is a 4-frame. Let $(\mathcal{L}^0, \mathcal{R}^0, \mathcal{L}^1, \mathcal{R}^1)$ be the partition of $\mathcal{E}_4$ defined as follows: For each $(u, v) \in \mathcal{E}_4$ let x be the node such that $v \in C_x$, and let y be the node (if it exists) closest to the root of $\mathcal{T}$ such that $(u, v) \in E_y$ and u is not a target vertex of F_y. If y exists, then (u, v) is incident to a corner c on F_y. If there is an $\alpha \in \{0, 1\}$ such that c is on the clockwise disegment of F_y between s_y^α and t_y^α we assign (u, v) to $\mathcal{R}^\alpha$. Otherwise there must be an $\alpha \in \{0, 1\}$ such that c is on the counterclockwise disegment of F_y between $s_y^{1-\alpha}$ and t_y^α, and we assign (u, v) to $\mathcal{L}^\alpha$. If no such y exists, (u, v) must be incident to t_x^α for some $\alpha \in \{0, 1\}$ and we (arbitrarily) assign (u, v) to $\mathcal{L}^\alpha$.

Lemma III.36. Let $(\mathcal{L}^0, \mathcal{R}^0, \mathcal{L}^1, \mathcal{R}^1)$ be the partition from Definition III.35. Then, for any $x \in \mathcal{T}$, $(E_x \cap \mathcal{L}^0, E_x \cap \mathcal{R}^0, E_x \cap \mathcal{L}^1, E_x \cap \mathcal{R}^1)$ is a partition of $E_x \cap \mathcal{E}_4$, such that for $\alpha \in \{0, 1\}$ all tails in $E_x \cap \mathcal{R}^\alpha$ are on the clockwise disegment of F_x between s^α and t^α, and all tails in $E_x \cap \mathcal{L}^\alpha$ are on the counterclockwise disegment of F_x between $s^{1-\alpha}$ and t^α.

Proof: Consider an edge $(u, v) \in \mathcal{E}_4$. If u is a target vertex in all frames F_x where $(u, v) \in E_x$, then by Definitions III.34 and III.35 there is an $\alpha \in \{0, 1\}$ such that (u, v) in $\mathcal{L}^\alpha$ and $t_x^\alpha = u$ for all such st-nodes x. Thus, for each such x, the tail of (u, v) is on the counterclockwise disegment of F_x between $s_x^{1-\alpha}$ and t_x^α as required.

Otherwise, there is an st-node y and an $\alpha \in \{0,1\}$, such that u is not a target of F_y, and for any ancestor z to y with $(u,v) \in E_z$, $u = t_z^\alpha$. By Definition III.35, we then have (u,v) in either $\mathcal{L}^\alpha$ or $\mathcal{R}^\alpha$, and t_z^α is on both the required segments. Thus, the statement holds for edges incident to a target of F_x.

Finally, let z and x be st-nodes such that z is the parent of x and $(u,v) \in E_z \cap E_x$, and u is not a target of F_z. Then, u is not a target of F_x, either, and:

- If u is on the clockwise disegment between s_z^α and t_z^α, then u is on the clockwise disegment between s_x^α and t_x^α.
- If u is on the counterclockwise disegment between $s_z^{1-\alpha}$ and t_z^α, then u is on the counterclockwise disegment between $s_x^{1-\alpha}$ and t_x^α.

Thus, since u is on the correct disegment of F_y, it will also be on the correct disegment of all descendants of y, and we are done. ■

Similar to the definition of $d_2[v]$; v's depth in $\mathcal{T}_2$, we may define the depth of v in the entire st-decomposition, $\mathcal{T}$. Given a vertex v belonging to some st-node $c[v]$, we let $j_2[v]$ denote the nearest ancestor to $c[v]$ whose frame is a 2-frame.

Definition III.37. Let $\mathcal{T}$ be an st-decomposition of $G = (V,E)$. For any vertex $v \in V$ define:

$$\begin{aligned} c[v] &:= \text{The node } x \text{ in } \mathcal{T} \text{ such that } v \in V[C_x] \\ d[v] &:= \text{The depth of } c[v] \text{ in } \mathcal{T} \\ J_2[v] &:= \{\text{depth}(x) |\ x \text{ is a non-root ancestor to } c[v] \text{ in } \mathcal{T} \text{ and } F_x \text{ is a 2-frame}\} \\ j_2[v] &:= \max(J_2[v])) \end{aligned}$$

The number $j_2[v]$ is especially useful for 4-frame nodes. On the path from the root to the component of v in the s-t-decomposition tree, there will be a last component whose frame is a 2-frame. We call the depth of the next component on the path $j_2[v]$. If $c[v]$ has a 4-frame, then for the rest of the path, that is, depth i with $j_2[v] \le i < d[v]$, we will have 4-frames nested in 4-frames, which gives a lot of useful structure.

Given a vertex v and a 4-frame F ancestral to v, we may now define the at most four best vertices that can reach v; one for each disegment of F. They will be called $l_i^0(v), r_i^0(v), l_i^1(v)$, and $r_i^1(v)$.

Definition III.38. For any $j_2[v] \le i < d[v]$ and $\alpha \in \{0,1\}$, let x be the ancestor of $c[v]$ at depth $i+1$ and define:

$$\begin{aligned} F_i(v) &:= F_x & E_i(v) &:= E_x \cap \mathcal{E}_4 \\ L_i^\alpha(v) &:= E_x \cap \mathcal{L}^\alpha & R_i^\alpha(v) &:= E_x \cap \mathcal{R}^\alpha \\ \widehat{L}_i^\alpha(v) &:= \left\{(w,w') \in L_i^\alpha(v) \middle|\ w' \rightsquigarrow v\right\} & \widehat{R}_i^\alpha(v) &:= \left\{(w,w') \in R_i^\alpha(v) \middle|\ w' \rightsquigarrow v\right\} \\ \widehat{F}_i(v) &:= \widehat{L}_i^0(v) \cup \widehat{R}_i^0(v) \cup \widehat{L}_i^1(v) \cup \widehat{R}_i^1(v) \end{aligned}$$

$$l_i^\alpha(v) := \begin{cases} \bot & \text{if } \widehat{L}_i^\alpha(v) = \emptyset \\ \text{the last vertex in } \text{tail}(\widehat{L}_i^\alpha(v)) \text{ on the counterclockwise dipath of } F_x & \text{otherwise} \end{cases}$$

$$r_i^\alpha(v) := \begin{cases} \bot & \text{if } \widehat{R}_i^\alpha(v) = \emptyset \\ \text{the last vertex in } \text{tail}(\widehat{R}_i^\alpha(v)) \text{ on the clockwise dipath of } F_x & \text{otherwise} \end{cases}$$

$$s_i^\alpha(v) := \text{The vertex associated with } s_x^\alpha \qquad t_i^\alpha(v) := \text{The vertex associated with } t_x^\alpha$$

Additionally, let $L_i^\alpha(v)$ and $\widehat{L}_i^\alpha(v)$ be totally ordered by the position of the starting vertices on the counterclockwise disegment of F_x and the clockwise order around each starting vertex. Similarly, let

$R_i^\alpha(v)$ and $\widehat{R}_i^\alpha(v)$ be totally ordered by the position of the starting vertices on the clockwise disegment of F_x and the counterclockwise order around each starting vertex.

We know from Section III-C that we can find the relevant vertices on each 2-frame surrounding v. The goal in this section is a data structure for efficiently computing $l_i^\alpha(v)$ and $r_i^\alpha(v)$ for $j_2[v] \le i < d[v]$.

As for 2-frames, the idea is to define some suitable trees that allow us to compute some related functions, and a crossing lemma that lets us use these to compute the functions we actually want.

Definition III.39. For any vertex $v \in V$ and $\alpha \in \{0,1\}$ let

$$p_l^\alpha[v] := \begin{cases} \bot & \text{if } d[v] = 0 \vee \ F_{d[v]-1}(v) \text{ is a 2-frame} \\ l^\alpha_{d[v]-1}(v) & \text{otherwise} \end{cases}$$

$$p_r^\alpha[v] := \begin{cases} \bot & \text{if } d[v] = 0 \vee \ F_{d[v]-1}(v) \text{ is a 2-frame} \\ r^\alpha_{d[v]-1}(v) & \text{otherwise} \end{cases}$$

and let T_l^α and T_r^α denote the rooted forests over V whose parent pointers are p_l^α and p_r^α respectively.

Definition III.40. For any $v \in V \cup \{\bot\}$, $\alpha \in \{0,1\}$, and $i \ge j_2[v]$ let

$$l_i'^\alpha(v) := \begin{cases} v & \text{if } v = \bot \vee d[v] \le i \\ l_i'^\alpha(p_l^\alpha[v]) & \text{otherwise} \end{cases} \qquad r_i'^\alpha(v) := \begin{cases} v & \text{if } v = \bot \vee d[v] \le i \\ r_i'^\alpha(p_r^\alpha[v]) & \text{otherwise} \end{cases}$$

Lemma III.41 (Crossing lemma)**.** Let $v \in V$, $\alpha \in \{0,1\}$, and $j_2[v] \le i < d[v] - 1$.

$$l_i^\alpha(v) \ne l_i'^\alpha(l_{i+1}^\alpha(v)) \implies l_i^\alpha(v) = l_i'^\alpha(m) \wedge r_i^\alpha(v) = r_i'^\alpha(m) \wedge d[m] = i+1$$
$$\text{where } m = r_{i+1}^\alpha(v) \ne \bot$$
$$r_i^\alpha(v) \ne r_i'^\alpha(r_{i+1}^\alpha(v)) \implies l_i^\alpha(v) = l_i'^\alpha(m) \wedge r_i^\alpha(v) = r_i'^\alpha(m) \wedge d[m] = i+1$$
$$\text{where } m = l_{i+1}^\alpha(v) \ne \bot$$

This Crossing Lemma (see Figure 7) is the 4-frame version of Lemma III.23. Again, the proof needs som additional technical lemmas, and is deferred to the end of this section.

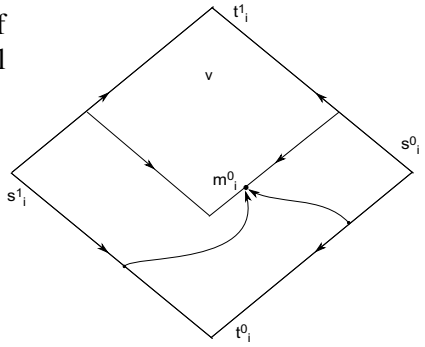

Figure 7. Sometimes, the best path from $L_i^0(v)$ to v must go through $R_{i+1}^0(v)$.

Definition III.42. Let $v \in V$, $\alpha \in \{0,1\}$, and $j_2[v] \le i < d[v]$.

$$m_i^\alpha(v) := \begin{cases} v & \text{if } i+1 = d[v] \\ l_{i+1}^\alpha(v) & \text{if } i+1 < d[v] \wedge r_i^\alpha(v) \ne r_i'^\alpha(r_{i+1}^\alpha(v)) \\ r_{i+1}^\alpha(v) & \text{if } i+1 < d[v] \wedge l_i^\alpha(v) \ne l_i'^\alpha(l_{i+1}^\alpha(v)) \\ m_{i+1}^\alpha(v) & \text{otherwise} \end{cases}$$

Corollary III.43. Let $v \in V$, $\alpha \in \{0,1\}$, and $j_2[v] \le i < d[v] - 1$. If $l_i^\alpha(v) \ne l_i'^\alpha(l_{i+1}^\alpha(v))$ or $r_i^\alpha(v) \ne r_i'^\alpha(r_{i+1}^\alpha(v))$ then

$$l_i^\alpha(v) = l_i'^\alpha(m_i^\alpha(v)) \qquad \wedge \qquad r_i^\alpha(v) = r_i'^\alpha(m_i^\alpha(v)) \qquad \wedge \qquad d[m_i^\alpha(v)] = i+1$$

Proof: This is just a reformulation of Lemma III.41 in terms of $m_i^\alpha(v)$. ∎

Lemma III.44. For any vertex $v \in V$, $\alpha \in \{0,1\}$, and $j_2[v] \le i < d[v]$

$$l_i^\alpha(v) = l_i'^\alpha(m_i^\alpha(v)) \qquad \wedge \qquad r_i^\alpha(v) = r_i'^\alpha(m_i^\alpha(v))$$

Definition III.45. For any vertex $v \in V$, and $\alpha \in \{0,1\}$ let

$$M^\alpha[v] := \left\{ i \middle|\, j_2[v] < i < d[v] \wedge m^\alpha_{i-1}(v) \neq m^\alpha_i(v) \right\}$$
$$p^\alpha_m[v] := \begin{cases} \bot & \text{if } M^\alpha[v] = \emptyset \\ m^\alpha_{\max M^\alpha[v]-1}(v) & \text{otherwise} \end{cases}$$

And define T^α_m as the rooted forest over V whose parent pointers are p^α_m.

Theorem III.46. *There exists a practical RAM data structure that for any good st-decomposition of a graph with n vertices uses $\mathcal{O}(n)$ words of $\mathcal{O}(\log n)$ bits and can answer $l^\alpha_i(v)$ and $r^\alpha_i(v)$ queries in constant time.*

Proof: For any vertex $v \in V$, and $\alpha \in \{0,1\}$ let

$$D^\alpha_l[v] := \{i|\ v \text{ has a proper ancestor } w \text{ in } T^\alpha_l \text{ with } d[w] = i\}$$
$$D^\alpha_r[v] := \{i|\ v \text{ has a proper ancestor } w \text{ in } T^\alpha_r \text{ with } d[w] = i\}$$

Now, store level ancestor structures for each of T^α_l, T^α_r, and T^α_m, together with $d[v]$, $j_2[v]$, $J_2[v]$, $D^\alpha_l[v]$, $D^\alpha_r[v]$, and $M^\alpha[v]$ for each vertex. Since the height of the st-decomposition is $\mathcal{O}(\log n)$ each of $J_2[v]$, $D^\alpha_l[v]$, $D^\alpha_r[v]$, and $M^\alpha[v]$ can be represented in a single $\mathcal{O}(\log n)$-bit word.

This representation allows us to find $d[m^\alpha_i(v)] = \operatorname{succ}(M^\alpha[v] \cup \{d[v]\}, i)$ in constant time, as well as computing the depth in T^α_m of $m^\alpha_i(v)$. Then using the level ancestor structure for T^α_m we can compute $m^\alpha_i(v)$ in constant time.

Similarly, this representation of the $D^\alpha_l[v]$ set lets us compute the depth in T^α_l of $l'^\alpha_i(v)$ in constant time, and, with the level ancestor structure, this lets us compute $l'^\alpha_i(v)$ in constant time. A symmetric argument shows that we can compute $r'^\alpha_i(v)$ in constant time.

Finally, lemma III.44 says we can compute $l^\alpha_i(v)$ and $r^\alpha_i(v)$ in constant time given constant-time functions for l', r', and m. ∎

Technical lemmas and proofs (4-frames): To prove the Lemmas III.41 and III.44, we prove some technical lemmas.

Lemma III.47. For any vertex $v \in V$ and $j_2[v] \leq i < d[v]$: $\widehat{F}_i(v) \neq \emptyset$

Proof: Let x be the ancestor of $c[v]$ at depth $i+1$. Since G is a single-source graph, there is a path from s to v. This path must contain a vertex in $V[F_x]$. But then the edge following the last such vertex on the path must be in $\widehat{L}^0_i(v) \cup \widehat{R}^0_i(v) \cup \widehat{L}^1_i(v) \cup \widehat{R}^1_i(v)$ which is therefore nonempty. ∎

Lemma III.48. For any $u, v \in V$, $j_2[v] \leq i < d[u]$, and $\alpha \in \{0,1\}$: If $u \rightsquigarrow v$ then $\widehat{L}^\alpha_i(u) \subseteq \widehat{L}^\alpha_i(v)$ and $\widehat{R}^\alpha_i(u) \subseteq \widehat{R}^\alpha_i(v)$.

Proof: Since $u \rightsquigarrow v$, $c[u]$ is ancestor to $c[v]$, and so, $L^\alpha_i(u) = L^\alpha_i(v)$, and hence, $\widehat{L}^\alpha_i(u) \subseteq \widehat{L}^\alpha_i(v)$. Similarly, $R^\alpha_i(u) = R^\alpha_i(v)$, and $\widehat{R}^\alpha_i(u) \subseteq \widehat{R}^\alpha_i(v)$. ∎

Lemma III.49. Given any vertex $v \in V$, $j_2[v] \leq i < d[v]$, $\alpha \in \{0,1\}$, and $(w, w') \in E_i(v)$. Then:

$$(w,w') \in \widehat{L}^\alpha_i(v) \quad \Longrightarrow \quad (w,w') \in \widehat{L}^\alpha_{i'}(v) \text{ for all } i', \max\{d[w], j_2[v]\} \leq i' < \min\left\{d[w'], d[v]\right\}$$
$$(w,w') \in \widehat{R}^\alpha_i(v) \quad \Longrightarrow \quad (w,w') \in \widehat{R}^\alpha_{i'}(v) \text{ for all } i', \max\{d[w], j_2[v]\} \leq i' < \min\left\{d[w'], d[v]\right\}$$

Proof: Let $j = \max\{d[w], j_2[v]\}$ and $k = \min\{d[w'], d[v]\}$. Clearly, $(w,w') \in E_{i'}(v)$ for all $j \leq i' < k$. Suppose $(w,w') \in \widehat{L}^\alpha_i(v) \subseteq L^\alpha_i(v)$, then, since $j \leq i < k$, the definition gives us $(w,w') \in L^\alpha_{i'}(v)$ for all $j \leq i' < k$. And since $w' \rightsquigarrow v$, this implies $(w,w') \in \widehat{L}^\alpha_{i'}(v)$ for all $j \leq i' < k$ and the result follows. The case for R is symmetric. ∎

Lemma III.50. Let $v \in V$, $\alpha \in \{0,1\}$, and $i \geq j_2[v]$ be given, then

$$\begin{array}{lllll} i = d[v]-1 & \implies & l_i'^{\alpha}(v) = l_i^{\alpha}(v) & \wedge & r_i'^{\alpha}(v) = r_i^{\alpha}(v) \\ i \leq d[v]-1 & \implies & l_i'^{\alpha}(v) \in \mathrm{tail}(\widehat{L}_i^{\alpha}(v)) \cup \{\bot\} & \wedge & r_i'^{\alpha}(v) \in \mathrm{tail}(\widehat{R}_i^{\alpha}(v)) \cup \{\bot\} \\ i > d[v]-1 & \implies & l_i'^{\alpha}(v) = v & \wedge & r_i'^{\alpha}(v) = v \end{array}$$

Proof: We will show this for l' only, as r' is completely symmetrical. If $i > d[v]-1$ then $d[v] \leq i$ and we get $l_i'^{\alpha}(v) = v$ directly from the definition of l'. Similarly, if $i = d[v]-1$, then $l_i'^{\alpha}(v) = l_i'^{\alpha}(p_l^{\alpha}[v]) = l_i'^{\alpha}(l_{d[v]-1}^{\alpha}(v)) = l_i'^{\alpha}(l_i^{\alpha}(v)) = l_i^{\alpha}(v) \in \mathrm{tail}(\widehat{L}_i^{\alpha}(v)) \cup \{\bot\}$. Finally, suppose $i < d[v]-1$. If $l_i'^{\alpha}(v) = \bot$ we are done, so suppose that is not the case. Let u be the child of $l_i'^{\alpha}(v)$ in T_l that is ancestor to v. Then $l_i'^{\alpha}(v) = l_i'^{\alpha}(u) = p_l^{\alpha}[u] = l_{d[u]-1}^{\alpha}(u)$. By definition of $l_{d[u]-1}^{\alpha}(u)$, there exists an edge $(w,w') \in \widehat{L}_{d[u]-1}^{\alpha}$ where $w = l_{d[u]-1}^{\alpha}(u)$ and $d[w] \leq i < d[w'] \leq d[u]$, and by setting $(v, i, (w,w')) = (u, d[u]-1, (w,w'))$ in Lemma III.49, we get $(w,w') \in \widehat{L}_i^{\alpha}(u)$, and therefore $l_i'^{\alpha}(v) \in \mathrm{tail}(\widehat{L}_i^{\alpha}(u))$. But since $u \rightsquigarrow v$ we have $\widehat{L}_i^{\alpha}(u) \subseteq \widehat{L}_i^{\alpha}(v)$ by Lemma III.48 and we are done. ∎

Lemma III.51. Let $v \in V$, $\alpha \in \{0,1\}$, and $j_2[v] \leq i \leq j$ then

$$l_i'^{\alpha}(l_j'^{\alpha}(v)) = l_i'^{\alpha}(v) \qquad \wedge \qquad r_i'^{\alpha}(r_j'^{\alpha}(v)) = r_i'^{\alpha}(v)$$

Proof: $l_j'^{\alpha}(v)$ is on the path from v to $l_i'^{\alpha}(v)$ in T_l, so this follows trivially from the recursion. The case for r' is symmetric. ∎

Lemma III.52. Let $v \in V$, $\alpha \in \{0,1\}$, and $j_2[v] \leq i < d[v]-1$, then

$$l_i^{\alpha}(v) = \bot \implies l_i'^{\alpha}(l_{i+1}^{\alpha}(v)) = \bot \qquad \wedge \qquad r_i^{\alpha}(v) = \bot \implies r_i'^{\alpha}(r_{i+1}^{\alpha}(v)) = \bot$$

Proof: If $l_i^{\alpha}(v) = \bot$ then $\widehat{L}_i^{\alpha}(v) = \emptyset$, so either $l_{i+1}^{\alpha}(v) = \bot$ implying $l_i'^{\alpha}(l_{i+1}^{\alpha}(v)) = \bot$ by the definition of l', or $l_{i+1}^{\alpha}(v) \notin \mathrm{tail}(\widehat{L}_i^{\alpha}(v))$ so $d[l_{i+1}^{\alpha}(v)] = i+1$ and by Lemma III.50 $l_i'^{\alpha}(l_{i+1}^{\alpha}(v)) \in \mathrm{tail}(\widehat{L}_i^{\alpha}(l_{i+1}^{\alpha}(v))) \cup \{\bot\} \subseteq \mathrm{tail}(\widehat{L}_i^{\alpha}(v)) \cup \{\bot\} = \{\bot\}$ so again $l_i'^{\alpha}(l_{i+1}^{\alpha}(v)) = \bot$. The case for r is symmetric. ∎

We may now prove Lemma III.41 (Crossing Lemma).

Proof of Lemma III.41 (Crossing Lemma): Suppose $l_i^{\alpha}(v) \neq l_i'^{\alpha}(l_{i+1}^{\alpha}(v))$ (the case $r_i^{\alpha}(v) \neq r_i'^{\alpha}(r_{i+1}^{\alpha}(v))$ is symmetrical). Then, $l_i^{\alpha}(v) \neq \bot$ by lemma III.52. Thus, there is a last edge $(w,w') \in \widehat{L}_i^{\alpha}(v)$ with $w = l_i^{\alpha}(v)$ and $d[w] \leq i < d[w']$ and a path $P = w' \rightsquigarrow v$.

Now, $(w,w') \notin E_{i+1}(v)$, since otherwise by Definition III.38 $(w,w') \in L_{i+1}^{\alpha}(v)$, and since $w' \rightsquigarrow v$, even $(w,w') \in \widehat{L}_{i+1}^{\alpha}(v)$, implying $l_i^{\alpha}(v) = l_{i+1}^{\alpha}(v)$, and thus, $l_i^{\alpha}(v) = l_i'^{\alpha}(l_{i+1}^{\alpha}(v))$ by Lemma III.50, contradicting our assumption.

Since $(w,w') \notin E_{i+1}(v)$, the path P must cross $\widehat{F}_{i+1}(v)$. Let (u,u') be the unique edge in $P \cap \widehat{F}_{i+1}(v)$. Then, $w' \rightsquigarrow u$ so $d[u] \geq i+1$ and $(u,u') \notin L_{i+1}^{\alpha}(v)$ since otherwise $d[l_{i+1}^{\alpha}(v)] = i+1$ and hence by Lemma III.50 $l_i^{\alpha}(v) = l_i'^{\alpha}(l_{i+1}^{\alpha}(v))$, again contradicting our assumption.

Also, $t_i^{\alpha}(v) \neq t_{i+1}^{\alpha}(v)$ because $t_i^{\alpha}(v) = t_{i+1}^{\alpha}(v)$ would imply $(w,w') \in L_{i+1}^{\alpha}(v) \cup \{\bot\}$ which we have just shown is not the case.

Since $t_i^{\alpha}(v) \neq t_{i+1}^{\alpha}(v)$, then, by definition, $t_i^{1-\alpha}(v) = t_{i+1}^{1-\alpha}(v)$ and hence $L_{i+1}^{1-\alpha}(v) \subseteq L_i^{1-\alpha}(v)$ and $R_{i+1}^{1-\alpha}(v) \subseteq R_i^{1-\alpha}(v)$, implying $d[w''] \leq i$ for all $w'' \in L_{i+1}^{1-\alpha}(v) \cup R_{i+1}^{1-\alpha}(v)$. Thus, $(u,u') \notin L_{i+1}^{1-\alpha}(v) \cup R_{i+1}^{1-\alpha}(v)$ since $d[u] > i$, and we can conclude that $(u,u') \in \widehat{R}_{i+1}^{\alpha}(v)$.

But then we can choose P so it goes through (m,m') where $m = r_{i+1}^{\alpha}(v) \neq \bot$. Now $i+1 \leq d[w'] \leq d[r_{i+1}^{\alpha}(v)] \leq i+1$ so $d[m] = i+1$.

Let e be the last edge in $\widehat{R}_i^{\alpha}(v)$ then any path $r_i^{\alpha}(v) \rightsquigarrow v$ that starts with e crosses $P \cup \widehat{R}_{i+1}^{\alpha}(v)$, implying that there exists such a path that contains (m,m') and thus $r_i^{\alpha}(v) = r_i^{\alpha}(m)$. Since $d[m] = i+1$, then $l_i^{\alpha}(v) = l_i'^{\alpha}(m)$ and $r_i^{\alpha}(v) = r_i'^{\alpha}(m)$ follows from lemma III.50. ∎

We may now prove the essential Lemma stating that $l_i^\alpha(v) = l_i'^\alpha(m_i^\alpha(v))$ (and similar for r_i^α):

Proof of Lemma III.44: The proof is by induction on j, the number of times the "otherwise" case is used before reaching one of the other cases when expanding the recursive definition of $m_i(v)$.

For $j = 0$, either $i + 1 = d[v]$ and the result follows from Lemma III.50, or $i + 1 < d[v]$ and $l_i(v) \neq l_i'(l_{i+1}(v))$ or $r_i(v) \neq r_i'(r_{i+1}(v))$. In either case we have by Corollary III.43, that $l_i^\alpha(v) = l_i'^\alpha(m_i^\alpha(v))$ and $r_i^\alpha(v) = r_i'^\alpha(m_i^\alpha(v))$.

For $j > 0$, we have $i + 1 < d[v]$ and $l_i(v) = l_i'(l_{i+1}(v))$ and $r_i(v) = r_i'(r_{i+1}(v))$ and $m_i(v) = m_{i+1}(v)$. By induction we can assume that $l_{i+1}^\alpha(v) = l_{i+1}'^\alpha(m_{i+1}^\alpha(v))$ and $r_{i+1}^\alpha(v) = r_{i+1}'^\alpha(m_{i+1}^\alpha(v))$. Then, by Lemma III.51, $l_i'^\alpha(l_{i+1}^\alpha(v)) = l_i'^\alpha(l_{i+1}'^\alpha(m_{i+1}^\alpha(v))) = l_i'^\alpha(m_{i+1}^\alpha(v)) = l_i'^\alpha(m_i^\alpha(v))$, showing that $l_i^\alpha(v) = l_i'^\alpha(m_i^\alpha(v))$ as desired. The case for r is symmetric. ■

IV. Acyclic planar In- out- graphs

For an in-out-graph, G, we have a source, s, that can reach all vertices of outdegree 0. Given such a source, s, we may assign all vertices a colour: A vertex is green if it can be reached from s, and red otherwise. We may also colour the directed edges: (u, v) has the same colour as its endpoints, or is a blue edge in the special case where u is red and v is green. Our idea is to keep the colouring and flip all non-green edges, thus obtaining a single source graph H with source s. (Any vertex was either green and thus already reachable from s, or could reach some target t, and is reachable from s in H via the first green vertex on its path to t.)

Consider the single source reachability data structure for the red-green graph, H. This alone does not suffice to determine reachability in G, but it does when endowed with a few extra words per vertex:

- M1 A red vertex u must remember the additional information of the best green vertices $BestGreen(u)$ on its own parent frame it can reach. There are at most 4 such vertices, one for each disegment.
- M2 Information about paths from a red to a green vertex in the same component. See Section IV-A.
- M3 Information about paths from a red vertex in some component C to a green vertex in an ancestor component of C. See Section IV-B.

Given a green vertex v, we know for each ancestral frame segment the best vertex that can reach v. For a red vertex u, given a segment p on an ancestral frame to u, we have information about the best vertex on p that may reach u in H via "ingoing" edges, that is, an edge from the corresponding $\widehat{F}_i(u)$. If that best vertex is red, then it is the best vertex on p that u can reach, again, from the "inside".

Now, $\text{reach}_G(u, v)$ has four cases, based on the colours of u and v:

- For green u and red v, $\text{reach}_G(u, v)$ = No.
- For green vertices u, v, $\text{reach}_G(u, v)$ = $\text{reach}_H(u, v)$
- For red vertices u, v, $\text{reach}_G(u, v)$ = $\text{reach}_H(v, u)$
- When u is red and v is green, to determine $\text{reach}_G(u, v)$ we need more work. It will depend on where in the hierarchy of components, u and v reside.

When u is red and v is green, a path from u to v must consist of a (possibly trivial) red path, a blue edge, and a (possibly trivial) green path. In the st-decomposition of H, red and blue edges can only either stay in an st-node, or go towards the root. Green edges, on the contrary, stay in an st-node, or go to a descendant. There are the following cases (see Figure 8) for $\text{reach}_G(u, v)$, based on where in the heirarchy of components u and v are.

1) $c[u] = c[v]$. There may be a path from u to v:
   - Staying within the st-node, that is, $\text{reach}_{c[u]}(u, v)$. To handle this case we need to store more information, see Section IV-A.
   - Via a green vertex w in the parent frame of u. For each candidate $w \in BestGreen(u)$, try $\text{reach}_H(w, v)$. (See M1).

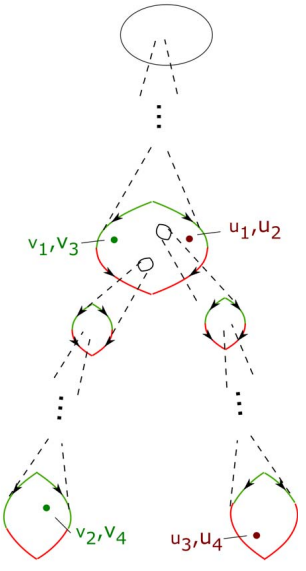

Figure 8. Cases 1 through 4.

2) $c[u] \prec c[v]$. There may be a path from u to v:
   - Via a green vertex w in the parent frame of u, $\text{reach}_H(w, v)$. (See M1).
   - Via a green vertex w, where $c[w] = c[u]$, then $\text{reach}_G(u, w)$ is in case 1 above. v can calculate the at most 4 such ws from the single source structure, namely $l(v)$ and $r(v)$, or $l^\alpha(v)$ and $r^\alpha(v)$.
3) $c[u] \succ c[v]$. There may be a path from u to v:
   - Via a red edge (w', w) in G with $c[w] \preceq c[v] \prec c[w'] \preceq c[u]$. That is, in the single-source structure for H, u can find its best vertex w for each disegment of the parent frame of $c[v]$. For a path via that disegment to exist, w must be red, and $\text{reach}_G(w, v)$, which is in case 1 or 2 above, must return true.
   - Via a blue edge (w', w) with $c[w] \preceq c[v] \prec c[w'] \preceq c[u]$. We handle this case in Section IV-B.
4) $c[u], c[v] \succ N$, where $N = \text{lca}(c[u], c[v])$. A path from u to v must go:
   - Via w, $c[w] \preceq N$, then $\text{reach}_G(u, w)$ is in case 3 above. v computes at most 4 such ws from the single source structure, and note that all the vertices that v computes must be green.

A. Intracomponental blue edges

Consider the set of "blue" edges (a, b) from G where both the red vertex a and green b reside in some given component in the s-t-decomposition of H.

Lemma IV.1. We may assign to each vertex ≤ 2 numbers, such that if red u remembers $i, j \in \mathbb{N}$ and green v remembers $l, r \in \mathbb{N}$, then u can reach v if and only if $i \leq l \leq j$ or $i \leq r \leq j$ or $\min\{l, r\} \leq j < i$ or $j < i \leq \max\{l, r\}$.

Proof: The key observation is that we may enumerate all blue edges $b_0 = (u_0, v_0), \ldots b_i = (u_m, v_m)$ such that any red vertex can reach a segment of their endpoints, $v_i, \ldots, v_j$. Namely, the blue edges form a minimal cut in the planar graph which separates the red from the green vertices, and this cut induces a cyclic order. In this order, each red vertex may reach a segment of blue edges, and each green vertex may reach a segment of blue edge endpoints. Thus, the blue edge endpoints reachable from a given red vertex (through any path) is a union of overlapping segments, which is again a segment.

Now each red vertex remembers the indices of the first v_i and last v_j blue edge endpoint it may reach. For a green vertex v, the s-t-subgraph with v as target has a delimiting face consisting of two paths, P and Q. v remembers the indices l, r of the latest blue edge endpoints $v_l \in P$ and $v_r \in Q$, if they exist. Clearly, if l or r is within range, u may reach v. Contrarily, if u may reach v, it must do so via some vertex v' on $P \cup Q$. But then v' must be able to reach v_l or v_r, and thus, l or r is within range. ■

B. Intercomponental blue edges

For any red vertex u, for a blue edge (u', v), where u' is reachable from u and separated from u by the frame $F_i(u)$, then one of u's "best" vertices, $l_i(u), r_i(u)$ or $l_i^\alpha(u), r_i^\alpha(u)$, is a red vertex, and can reach u'. For any red vertex w, let $E_{c[w]}$ denote the edges of w's parent frame, $F_{c[w]}$. Consider the set $B(w)$ of those blue edges in $E_{c[w]}$ that are reachable from w in G (or, equvalently, can reach w in H). For each disegment of $F_{c[w]}$, there is at most one "best" edge of $B(w)$, that is, whose green head is closest to the source. Let each red vertex remember the best ≤ 4 blue edges it can reach on its own frame. Then we can define 4 bitmasks $\{B^\beta(u)\}_{0 \leq \beta \leq 3}$ such that for any i finding the highest 1-bit $\leq i$ in each, gives at most 4 levels such that u's best vertices on those levels together know the best blue edges for u.

ACKNOWLEDGEMENT

Mikkel Thorup's and Jacob Holm's research is partly supported by Mikkel Thorup's Advanced Grant DFF-0602-02499B from the Danish Council for Independent Research under the Sapere Aude research career programme.

REFERENCES

[1] ALSTRUP, S., HOLM, J., LICHTENBERG, K. D., AND THORUP, M. Maintaining information in fully dynamic trees with top trees. *ACM Trans. Algorithms 1*, 2 (Oct. 2005), 243–264.

[2] ALSTRUP, S., SECHER, J. P., AND SPORK, M. Optimal on-line decremental connectivity in trees. *Inf. Process. Lett. 64* (1997), 161–164.

[3] ARIKATI, S., CHEN, D., CHEW, L., DAS, G., SMID, M., AND ZAROLIAGIS, C. Planar spanners and approximate shortest path queries among obstacles in the plane. In *ESA '96* (1996), pp. 514–528.

[4] CHEN, D., AND XU, J. Shortest path queries in planar graphs. In *STOC '00* (2000), pp. 469–478.

[5] DJIDJEV, H. Efficient algorithms for shortest path queries in planar digraphs. In *WG '96* (1996), pp. 151–165.

[6] DJIDJEV, H., PANZIOU, G., AND ZAROLIAGIS, C. Computing shortest paths and distances in planar graphs. In *ICALP '91* (1991), pp. 327–339.

[7] DJIDJEV, H., PANZIOU, G., AND ZAROLIAGIS, C. Fast algorithms for maintaining shortest paths in outerplanar and planar digraphs. In *FCT '95* (1995), pp. 191–200.

[8] HAREL, D., AND TARJAN, R. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput. 13*, 2 (1984), 338–355.

[9] KAMEDA, T. On the vector representation of the reachability in planar directed graphs. *Inf. Process. Lett. 3*, 3 (1975), 75–77.

[10] KAWARABAYASHI, K., KLEIN, P., AND SOMMER, C. Linear-space approximate distance oracles for planar, bounded-genus, and minor-free graphs. In *ALP '11* (2011), pp. 135–146.

[11] KAWARABAYASHI, K., SOMMER, C., AND THORUP, M. More compact oracles for approximate distances in undirected planar graphs. In *SODA '13* (2013), pp. 550–563.

[12] KERNIGHAN, B., AND RITCHIE, D. *The C Programming Language*, 2nd ed. Prentice Hall, 1988.

[13] KLEIN, P. Preprocessing an undirected planar network to enable fast approximate distance queries. In *SODA '02* (2002), pp. 820–827.

[14] MILTERSEN, P. B. Lower bounds for static dictionaries on rams with bit operations but no multiplication. In *ICALP '96*. 1996, pp. 442–453.

[15] MOZES, S., AND SOMMER, C. Exact distance oracles for planar graphs. In *SODA '12* (2012), pp. 209–222.

[16] PĂTRAŞCU, M. Unifying the landscape of cell-probe lower bounds. *SIAM J. Comput. 40*, 3 (2011), 827–847. Announced at FOCS'08. See also arXiv:1010.3783.

[17] TAMASSIA, R., AND TOLLIS, I. A unified approach to visibility representations of planar graphs. *Disc. Comput. Geom. 1*, 1 (1986), 321–341.

[18] TAMASSIA, R., AND TOLLIS, I. Dynamic reachability in planar digraphs with one source and one sink. *Theor. Comput. Sci. 119*, 2 (1993), 331–343.

[19] TARJAN, R. Depth first search and linear graph algorithms. *SIAM J. Comput.* (1972).

[20] THORUP, M. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM 51*, 6 (2004), 993–1024.

[21] THORUP, M., AND ZWICK, U. Approximate distance oracles. *J. ACM 52*, 1 (2005), 183–192. Announced at STOC'01.